Advanced Control Systems Lab

ACSL TurtleBot3 e-Manual

June 2020

Contents

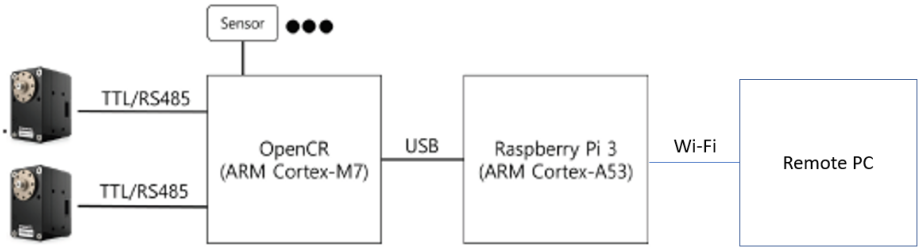
Abstract

This guide documents the changes made to TurtleBot3 software to meet the requirements of ACSL research group. It contains instructions on setting up the robot and code documentation.

1 Overview and Setup

TurtleBot3 is a small ROS-based mobile robot. The TurtleBot3 can be customized by changing the source code and addition of new hardware. TurtleBot3 Waffle Pi is equipped with a Raspberry Pi Camera Module (v2), a 360-degree Laser Distance Sensor (LDS) and Dynamixel XM430 210-T motors.

TurtleBot3 architecture is described in the following image.



The four major components of TurtleBot3 are

1. Raspberry Pi 3B+ Single Board Computer (SBC).
2. OpenCR Embedded System Board.
3. LDS and Camera Sensors.
4. Dynamixel Actuators.

Raspberry Pi has a WiFi module built onto the board which connects to a computer, known as the Remote PC. Remote PC is used to send commands for SLAM and Navigation to TB3 and visualization of the sensor data is also performed on the remote PC.

There is a one time setup that needs to be performed on the Remote PC, Raspberry Pi and OpenCR board. Most of the software is located in the form of a binary file that is burnt to the EEPROM of the STM32F746 chip on the OpenCR board. As the software used for ACSL projects is custom, it is burnt using Arduino IDE.

1.1 Remote PC Setup

To begin you will need to install the appropriate ROS version on your remote PC. The remote PC will be running ROS Melodic (EOL date: May 2023). The single board computer on Turtlebot3 will be running ROS Kinetic (EOL date: April 2021) for the ease of setup since ROS Melodic requires manual compilation when using Rasbian. No problem has yet been found by us on the communication between these 2 releases.

The following page from ros.org describes the process for adding and installing all necessary packages to your ROS Environment.

`http://wiki.ros.org/melodic/Installation/Ubuntu`

Note that please do not follow the Turtlebot3 guide to configure the ROS environment on the remote PC **(Step 6.1.1-6.1.3)**. Instead, simply follow the instructions in the link above.

Then, install required dependencies

```
sudo apt install ros-melodic-joy ros-melodic-teleop-twist-joy ros-melodic-teleop-twist-keyboard
    ros-melodic-laser-proc ros-melodic-rgbd-launch ros-melodic-depthimage-to-laserscan
    ros-melodic-rosserial-arduino ros-melodic-rosserial-python ros-melodic-rosserial-server
    ros-melodic-rosserial-client ros-melodic-rosserial-msgs ros-melodic-amcl ros-melodic-map-server
    ros-melodic-move-base ros-melodic-urdf ros-melodic-xacro ros-melodic-compressed-image-transport
    ros-melodic-rqt-image-view ros-melodic-gmapping ros-melodic-navigation ros-melodic-interactive-markers
```

Warning! We have used other dependencies but we didn't keep documentation, we need to figure out all the deps that we have used.

The next step is to configure network settings for permanent use. The Remote PC will be controlling the turtlebot pc via wifi using SSH protocol. Steps for configuring the wifi on the remote pc can be found here.

`https://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#network-configuration`

For this project we will use the Linksys Router on the **Linksys04294** network. It is helpful to write down the IP address of the remote pc, since you will need it later to setup the turtlebot pc network settings.

Then, clone and build the git repository.

```
git clone https://github.com/hanyiabc/ASCL_turtlebot3.git
cd ASCL_turtlebot3/
catkin_make
```

Add this environment variable to .bashrc and refresh the environment variables

```
echo "export␣TURTLEBOT3_MODEL=waffle_pi_effort_controller" >> ~/.bashrc
source ~/.bashrc
```

On the original Turtlebot code, this environment variable is used to define the model of the Turtlebot, possible values are **waffle, waffle_pi, burger**. Difference URDF file is loaded based on this environment variable. For this project however, we defined new URDF files for effort control, so we need to set the environment variable to what we have defined as the name of the model for effort control.

1.2 Rasberry Pi SBC Setup

Now it is time to look at the turtlebot SBC. For the turtlebot 3 Waffle Pi model, the on board PC is a raspberry PI. This step requires a micro SD card with adapter, a monitor with an HDMI input, a USB keyboard and mouse, and a power source for the turtlebot. ROBOTIS provides a prebuilt desktop environment for the turtlebot with ROS kinetic. Instructions for flashing this distribution can be found here (**Step 6.2.1.2 Install Linux Based on Raspbian, Do not use the other two methods.**).
`https://emanual.robotis.com/docs/en/platform/turtlebot3/raspberry_pi_3_setup/#raspberry-pi-3-setup`

To use the Raspberry PI without a physical access, use SSH. Example SSH command:

```
ssh pi@YOUR_RASPBERRY_PI_IP
```

For the ease of use, VNC configuration is recommended. This provide remote desktop funtionality. In case of Debian Stretch (the version that the manual will use), install Real VNC by running this command on the Raspberry PI:

```
sudo apt update
sudo apt install realvnc-vnc-server realvnc-vnc-viewer
```

THen enable VNC server by

```
sudo raspi-config
```

Then navigate to Interfacing Options. Scroll down and select VNC - Yes.

Ubuntu Desktop 18.04 comes with VNC client Remmina. On Windows, use VNC viewer `https://www.realvnc.com/en/connect/download/viewer/`

1.3 OpenCR Setup

The original source code for TurtleBot3 used the inbuilt PID controller of the Dynamixel XM430 210-T actuators. However, to benchmark the performance of the controllers, the control is shifted from the motors to higher level software such as Simulink or a custom ROS node.

The source code for OpenCR can be downloaded by cloning the following repository.

`https://github.com/hanyiabc/ASCL_turtlebot3.git`

Arduino IDE can be used to burn the **turtlebot3_core.ino** file located in **ACSL_turtlebot3/src/turtlebot3_core** directory. Instructions for setting up the Arduino IDE for TurtleBot3 can be found in the following link.

`https://emanual.robotis.com/docs/en/parts/controller/opencr10/#arduino-ide`

2 Repository Introduction

The repository is a catkin workspace. A catkin workspace has a src folder that contains multiple packages. The worth-mentio ning ones are:

- hls_lfcd_lds_driver
- raspicam_node
- ros_control
- simulink
- turtlebot3
  - turtlebot3_bringup
  - turtlebot3_navigation
  - turtlebot3_slam
  - turtlebot3_teleop
- turtlebot3_core
- turtlebot3_setup_motor
- turtlebot3_simulations

Not all packages are catkin package. **simulink**, **turtlebot3_core turtlebot3_setup_motor** are all just directories. All the rests are properly configured as catkin packages, meaning all the nodes defined under the package will be compiled when running the **catkain_make** command.

hls_lfcd_lds_driver and **raspicam_node** are drivers for the PI Camera and lidar. These will be running in the single board computer.

ros_control contains all the configuration and nodes necessary to control the robot. It contains the PID controller configurationl, PID gains, etc. Two launch files are included for controlling the robot in either the simulations or in the physical world. They are:

- turtlebot3_control.launch
- turtlebot3_control_simulation.launch

These launch file are not ran directly, instead they are inlcuded by other launch file that we will introduce later. The PID gains can be tuned by either changing the parameters in the launch file or use a ros package called rqt_reconfigure to adjust parameters with sliders at runtime. There are 3 nodes under the **ros_control** package. They are **differential_driver.py**, **ros_control_odom_pub** and **ros_control_node**. The **differential_driver.py** is a Python ros node that split the incoming **/cmd_vel** messages into left and right velocity for the differential drive robot based on vehicle geometry. The **cmd_vel** is the standard topic that ROS nav stack used to output the velocity command. It commands the robot in terms of translational velocity and orientation. The **ros_control_node** is a C++ ros node compiled from the soruce file: **message_redirect.cpp**. This node simply takes the left and right wheel angular velocities feedbacks from **joint_states** converts them to linear velocities and publish them into 2 Float64 messages for the PID controller to read. The **ros_control_odom_pub** is a C++ ros node compiled from the source file: **odom_publisher.cpp**.

The directory **simulink** contains all the MATLAB scrips and Simulink models for controlling the robot with MATLAB (currently not uisng MATLAB).

turtlebot3_bringup contains all the launch file for bringing up the Turtlebot in the SBC and the remote PC. **turtlebot3_physical_nav_control.launch** was added to the bringup package to provide a convinient one-file launch that will handles everything. This launch file brings up the turtlebot on the remote PC, runs the velocity control and the navigation stack with SLAM. A simulation version will be provided in the future.

turtlebot3_description contains the URDF and Gazebo descripion of the turtlebot for both simulation and physical robot.

For Waffle Pi, **turtlebot3_waffle_pi.gazebo.xacro** and **turtlebot3_waffle_pi.urdf.xacro** are provided by the turtlebot package, however, for effort control, we added 2 more files. They are:
turtlebot3_waffle_pi_effort_controller.gazebo.xacro and **turtlebot3_waffle_pi_effort_controller.urdf.xacro**

These files are derived from the original descriptions and added ability to control effort in both the simulation and the physical robot.

turtlebot3_navigation contains the default configuration for running the ROS navigation stack with the Turtlebot. A new launch file **turtlebot3_navigation_no_map.launch** was created. This launch file derived from the original navigation launch file, but the difference is that this configuration doesn't need a SLAM map for localizatin. The localizatin is provide by SLAM instaed of Adaptive Monte Carlo Localizatin. The default launch file **turtlebot3_navigation.launch** will run navigation assuming that a SLAM map has been drawn by running SLAM before. In this configuration, localizatin is provided by AMCL.

3 Torque Control using Dynamixel

Dynamixel is a microcontroller based actuator with in-built PID controller. OpenCR communicates with Dynamixel using packet communication via TTL/RS45 ports.

Hardware level abstraction is achieved via Dynamixel SDK library available in several programming languages. (CPP used in our case.) Dynamixel register addresses and byte sizes for both RAM and EEPROM memory provided in control table. RAM is most frequently used memory for robot applications, while startup settings are stored on EEPROM.

The Control Table is a structure that consists of multiple Data fields to store status or to control the device. Users can check current status of the device by reading a specific Data from the Control Table with Read Instruction Packets. WRITE Instruction Packets enable users to control the device by changing specific Data in the Control Table. Packet sizes range from 1 – 4 bytes.

Following is a snapshot of the Dynamixel EEPROM Control Table. Each data in the Control Table is restored to initial values when the device is turned on. Default values in the EEPROM area (addresses 0-63) are initial values of the device (factory default settings). If any values in the EEPROM area are modified by a user, modified values will be restored as initial values when the device is turned on. Initial Values in the RAM area are restored when the device is turned on.

Address	Size (Byte)	Data Name	Access	Default Value	Range	Unit
0	2	Model Number	R	1,030	-	-
2	4	Model Information	R	-	-	-
6	1	Firmware Version	R	-	-	-
7	1	ID	RW	1	0 ~ 253	-
8	1	Baud Rate	RW	1	0 ~ 7	-
9	1	Return Delay Time	RW	250	0 ~ 254	2 [µsec]
10	1	Drive Mode	RW	0	0 ~ 1	-
11	1	Operating Mode	RW	3	0 ~ 16	-
12	1	Secondary(Shadow) ID	RW	255	0 ~ 252	-
13	1	Protocol Type	RW	2	1 ~ 2	-
20	4	Homing Offset	RW	0	-1,044,479 ~ 1,044,479	1 [pulse]
24	4	Moving Threshold	RW	10	0 ~ 1,023	0.229 [rev/min]
31	1	Temperature Limit	RW	80	0 ~ 100	1 [°C]
32	2	Max Voltage Limit	RW	160	95 ~ 160	0.1 [V]
34	2	Min Voltage Limit	RW	95	95 ~ 160	0.1 [V]
36	2	PWM Limit	RW	885	0 ~ 885	0.113 [%]
38	2	Current Limit	RW	1,193	0 ~ 1,193	2.69 [mA]
44	4	Velocity Limit	RW	330	0 ~ 1,023	0.229 [rev/min]

For the purpose of torque control, the operating mode of the dynamixel motor needs to be set to **MODE 0**. This can be done using the motor setup code in the git repository. To change the operating mode to **MODE 0**, flash the turtlebot3_setup_motor firmware to the OpenCR board while the motors are connected to the board. Open up a serial terminal with Arduino or Terraterm, under the serial termianl, you will see the following options:

```
    1. setup left  motor
    2. setup right motor
    3. test  left  motor
    4. test  right motor
    5. setup left  motor for current control
    6. setup right  motor for current control
    7. test left  motor for current control
    8. test right  motor for current control
>>
```

The first 4 options comes with the OpenCR Arduino library. The last 4 options are added based on the first 4 options and they configure the left and right motors for torque control. For now, the current is limited to half of the highest value to avoid damage to the hardware. If changes are needed, the source code needs to be modified and the motors has to be setup again. To setup the left motor, simply put 5 in the termianl and press enter. Wait until it says it's finished. Then use option 6 for the right motor. After the setup is done, use option 7 and 8 to test the motor. The test option apply a small torque to the motor for one second and stops for one second. Make sure both motors are tested then, flash the turtlebot3_core firmware back. Once the operating mode is set. The OpenCR code is changed to write torque values to the Dynamixel RAM addresses, when the robot is live. The addresses that are useful for this purpose are given in the image below.

102	2	Goal Current	RW	-	-Current Limit(38) ~ Current Limit(38)	2.69 [mA]
104	4	Goal Velocity	RW	-	-Velocity Limit(44) ~ Velocity Limit(44)	0.229 [rev/min]
108	4	Profile Acceleration	RW	0	0 ~ 32,767 0 ~ 32,737	214.577 [rev/min^2] 1 [ms]
112	4	Profile Velocity	RW	0	0 ~ 32,767	0.229 [rev/min]
116	4	Goal Position	RW	-	Min Position Limit(52) ~ Max Position Limit(48)	1 [pulse]
120	2	Realtime Tick	R	-	0 ~ 32,767	1 [msec]
122	1	Moving	R	0	0 ~ 1	-
123	1	Moving Status	R	0	-	-
124	2	Present PWM	R	-	-	-
126	2	Present Current	R	-	-	2.69 [mA]
128	4	Present Velocity	R	-	-	0.229 [rev/min]
132	4	Present Position	R	-	-	1 [pulse]

The code that change the operating mode of the motor is below.

```
bool setupMotorLeftCurrentControl(void)
{
  CMD_SERIAL.println("Setup Motor Left for current control...");


  if (tb3_id < 0)
  {
    CMD_SERIAL.println("  no dxl motors");
  }
  else
  {
    write(portHandler, packetHandler2, tb3_id, 64, 1, 0);
    write(portHandler, packetHandler2, tb3_id, 7, 1, 1);
    tb3_id = 1;
    write(portHandler, packetHandler2, tb3_id, 8, 1, 3);
    portHandler->setBaudRate(1000000);
    write(portHandler, packetHandler2, tb3_id, 10, 1, 0);
    write(portHandler, packetHandler2, tb3_id, 11, 1, 0);

    // this set the limit to half
    write(portHandler, packetHandler2, tb3_id, 38, 2, 596);
    CMD_SERIAL.println("  Warning: Current limit is set to half to protect the motor!");
    CMD_SERIAL.println("  ok");
  }
}
```

```
bool setupMotorRightCurrentControl(void)
{
  CMD_SERIAL.println("Setup Motor Right for current control...");


  if (tb3_id < 0)
  {
    CMD_SERIAL.println("  no dxl motors");
  }
  else
  {
    write(portHandler, packetHandler2, tb3_id, 64, 1, 0);
    write(portHandler, packetHandler2, tb3_id, 7, 1, 2);
    tb3_id = 2;
    write(portHandler, packetHandler2, tb3_id, 8, 1, 3);
    portHandler->setBaudRate(1000000);
    write(portHandler, packetHandler2, tb3_id, 10, 1, 1);
    write(portHandler, packetHandler2, tb3_id, 11, 1, 0);

     // this set the limit to half
    write(portHandler, packetHandler2, tb3_id, 38, 2, 596);
    CMD_SERIAL.println("  Warning: Current limit is set to half to protect the motor!");
    CMD_SERIAL.println("  ok");
  }
}
```

The code used to test the torque control after setup is below.

```
void testMotorCurrentControl(uint8_t id)
{
  uint32_t pre_time;
  uint8_t toggle = 0;

  if (id == 1)
  {
    CMD_SERIAL.printf("Test Motor Left...");
  }
  else
  {
    CMD_SERIAL.printf("Test Motor Right...");
  }
  // We run at 1000000
  portHandler->setBaudRate(1000000);

  uint16_t model_number;
  int dxl_comm_result = packetHandler2->ping(portHandler, id, &model_number);
  if (dxl_comm_result == COMM_SUCCESS)
  {
    CMD_SERIAL.printf(" found type: %d\n", model_number);
    write(portHandler, packetHandler2, id, 64, 1, 1);

    toggle = 0;
    pre_time = millis();
    write(portHandler, packetHandler2, id, 102, 2, 100);
    while (1)
    {
      if (CMD_SERIAL.available())
      {
        flushCmd();
        break;
      }
```

```
      if (millis() - pre_time > 1000)
      {
        pre_time = millis();

        toggle ^= 1;

        if (toggle)
        {
          write(portHandler, packetHandler2, id, 102, 2, 0);
        }
        else
        {
          write(portHandler, packetHandler2, id, 102, 2, 100);
        }
      }
    }
    write(portHandler, packetHandler2, id, 102, 2, 0);
  }
  else
  {
    CMD_SERIAL.printf("  dxl motor ID:%d not found\n", id);
  }
}
```

The OpenCR Arduino library comes with a TurtleBot3MotorDriver class that helps controlling the 2 motors by commanding velocities. A new class TurtleBot3MotorTorqueDriver was derived (strictly speaking, not inherited, instead just copied and pasted) from the TurtleBot3MotorDriver with new member functions added for controlling the torque instead of velocities. The class is defined as below

---

```
class TurtleBot3MotorTorqueDriver
{
public:
    TurtleBot3MotorTorqueDriver();
    ~TurtleBot3MotorTorqueDriver();
    bool init(String turtlebot3);
    void close(void);
    bool setTorque(bool onoff);
    bool getTorque();
    bool readEncoder(int32_t &left_value, int32_t &right_value);
    bool writeVelocity(int64_t left_value, int64_t right_value);

    bool writeTorque(int16_t left_value, int16_t right_value);
    bool readTorque(float &left_torque, float &right_torque);
    bool controlMotor(const float wheel_radius, const float wheel_separation, float *value);
    bool controlMotor(float *torque);


private:
    uint32_t baudrate_;
    float protocol_version_;
    uint8_t left_wheel_id_;
    uint8_t right_wheel_id_;
    bool torque_;

    uint16_t dynamixel_limit_max_velocity_;
    uint16_t dynamixel_limit_max_current_;

    dynamixel::PortHandler *portHandler_;
    dynamixel::PacketHandler *packetHandler_;
```

```
    dynamixel::GroupSyncWrite *groupSyncWriteVelocity_;
    dynamixel::GroupSyncRead *groupSyncReadEncoder_;

    dynamixel::GroupSyncWrite *groupSyncWriteTorque_;
    dynamixel::GroupSyncRead *groupSyncReadTorque_;
```

The following member functions is added to the TurtleBot3MotorDriver class, which takes in input torque value and converts it into appropriate register values for writing to the appropriate addresses.

```
bool TurtleBot3MotorTorqueDriver::controlMotor(float *torque)
{
       bool dxl_comm_result = false;

       int16_t wheel_current_cmd[2];
       wheel_current_cmd[LEFT] = CURRENT_TO_OUTPUT(TORQUE_TO_CURRENT(torque[LEFT]));
       wheel_current_cmd[RIGHT] = CURRENT_TO_OUTPUT(TORQUE_TO_CURRENT(torque[RIGHT]));

       wheel_current_cmd[LEFT] = constrain(wheel_current_cmd[LEFT], -dynamixel_limit_max_current_,
           dynamixel_limit_max_current_);
       wheel_current_cmd[RIGHT] = constrain(wheel_current_cmd[RIGHT], -dynamixel_limit_max_current_,
           dynamixel_limit_max_current_);

       dxl_comm_result = writeTorque((int16_t)wheel_current_cmd[LEFT], (int16_t)wheel_current_cmd[RIGHT]);

       if (dxl_comm_result == false)
          return false;

       return true;
```

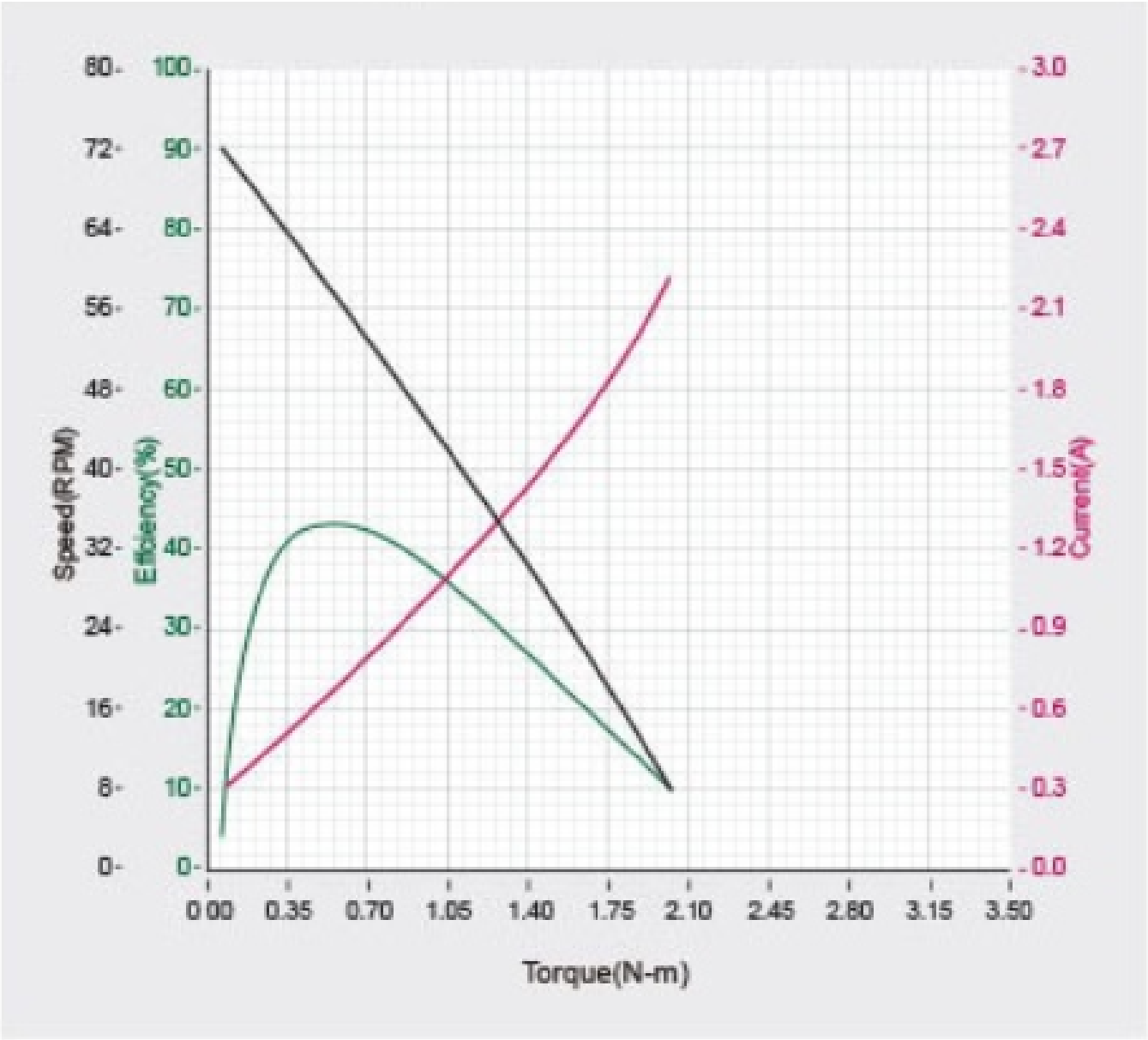
CURRENT_TO_OUTPUT and **TORQUE_TO_CURRENT** are macro functions that does the conversions from torque to current and current to a 2 byte value that the microcontroller undertand. All the necessary macros are defined as

```
#include <stdint.h>
#define WAFFLE_DXL_LIMIT_MAX_CURRENT 780
#define ADDR_X_GOAL_CURRENT        102
#define ADDR_X_PRESENT_CURRENT     126
#define LEN_X_GOAL_CURRENT         2
#define LEN_X_PRESENT_CURRENT      2

#define MAX_CURRENT_11V1             2.1
#define MAX_TORQUE_11V1              2.7

#define CURRENT_GOAL_UNIT            2.69
#define TORQUE_TO_CURRENT(t)         t * (MAX_CURRENT_11V1 / MAX_TORQUE_11V1)// convert torque to current
    in amp
#define CURRENT_TO_OUTPUT(a)         (int16_t)(a * 1000 / CURRENT_GOAL_UNIT)
#define CURRENT_TO_TORQUE(t)     t / (MAX_CURRENT_11V1 / MAX_TORQUE_11V1)// convert current in amp to
    torque in N-m
```

This conversion is based on a linear approximation of the actuator performance graph.



This function below takes the converted value and write to both motors using the Dynamixel SDK APIs.

```
bool TurtleBot3MotorTorqueDriver::writeTorque(int16_t left_value, int16_t right_value)
{
    bool dxl_addparam_result;
    int8_t dxl_comm_result;

    dxl_addparam_result = groupSyncWriteTorque_->addParam(left_wheel_id_, (uint8_t *)&left_value);
    if (dxl_addparam_result != true)
        return false;

    dxl_addparam_result = groupSyncWriteTorque_->addParam(right_wheel_id_, (uint8_t *)&right_value);
    if (dxl_addparam_result != true)
        return false;

    dxl_comm_result = groupSyncWriteTorque_->txPacket();
    if (dxl_comm_result != COMM_SUCCESS)
    {
        Serial.println(packetHandler_->getTxRxResult(dxl_comm_result));
        return false;
```

```
    }

    groupSyncWriteTorque_->clearParam();
    return true;
}
```

Subscribers are added to subscribe to the topics left_torque right_torque. The following use the function defined above and write the correspounding values to the correct memory location of the Dynamixel motor controller when new message comes in. When there is no new message, it will timeout and write zeros to both motors.

```
  if ((t-tTime[0]) >= (1000 / CONTROL_MOTOR_TORQUE_FREQUENCY))
  {

    updateGoalTorque();
    //this timeout will stop the motor if no message comes in

    if ((t-tTime[6]) > CONTROL_MOTOR_TIMEOUT)
    {
      motor_driver.controlMotor(zero_torque);
    }
    else {
      motor_driver.controlMotor(torque);
    }
```

This Python node will convert the cmd_vel to the left and right velocities command based on the robot geometry.

```
#!/usr/bin/env python2
import rospy

from std_msgs.msg import Float64
from sensor_msgs.msg import Imu, JointState
from math import sin,cos,atan2,sqrt,fabs
from geometry_msgs.msg import Twist

class Driver:
    def __init__(self):
        rospy.init_node('differential_driver')

        self._rate = rospy.get_param('~rate', 500)
        self._wheel_base = rospy.get_param('~wheel_base', 0.28)

        #self._left_speed_percent = 0
        #self._right_speed_percent = 0
        self._left_speed = 0
        self._right_speed = 0

        rospy.Subscriber('cmd_vel', Twist, self.velocity_received_callback)
        self.left_wheel_vel = rospy.Publisher('/left_vel', Float64, queue_size=100)
        self.right_wheel_vel = rospy.Publisher('/right_vel', Float64, queue_size=100)

    def velocity_received_callback(self, message):
        linear = message.linear.x
        angular = message.angular.z

        self.left_speed = linear - angular * self._wheel_base/2
        self.right_speed = linear + angular * self._wheel_base/2

        # print("left wheel velocity:",self.left_speed,"right wheel velocity:",self.right_speed)
        self.publish_velocity()

    def publish_velocity(self):
        self.left_wheel_vel.publish(self.left_speed)
```

```
        self.right_wheel_vel.publish(self.right_speed)

    def run(self):
        rate = rospy.Rate(self._rate)
        while not rospy.is_shutdown():
            rate.sleep()

if __name__ == '__main__':
    driver = Driver()
    driver.run()
```

The original Odometry publisher is in the microcontroller. Since the built-in differential drive plugin which provides Odometry that's used in the simulation was replaced by our own controller, An Odometry publisher was added to the system to provide the Odometry in the simulation only. The source code is below.

```
#include <ros/ros.h>
#include <ros/console.h>
#include <sensor_msgs/JointState.h>
#include <sensor_msgs/Imu.h>
#include <math.h>
#include <std_msgs/Float64.h>
#include <tf/transform_broadcaster.h>
#include <geometry_msgs/Quaternion.h>
#include <nav_msgs/Odometry.h>
#include <vector>
#include <string>
#include <iostream>
#include <chrono>
#include <thread>

#define WHEEL_RADIUS                0.033


using std::cout;
using std::endl;

sensor_msgs::Imu imu;
sensor_msgs::JointState lastJointState;
sensor_msgs::JointState currJointState = lastJointState;
geometry_msgs::Quaternion q;
nav_msgs::Odometry odom;
ros::Publisher odom_pub;
geometry_msgs::TransformStamped odom_tf;
std::string odom_header_frame_id;
std::string odom_child_frame_id;
double orientation[4];
double odom_pose[3];
double odom_vel[3];

void init(ros::NodeHandle n);
bool calcOdometry();
void updateOdometry();

bool calcOdometry()
{

  double wheel_l, wheel_r;   // rotation value of void updateOdometry(void)wheel [rad]
  double delta_s, theta, delta_theta;
  static double last_theta = 0.0;
  static double lastTime = ros::Time::now().toSec();
```

```
  double currTime;
  double v, w;              // v = translational velocity [m/s], w = rotational velocity [rad/s]
  double step_time;

  wheel_l = wheel_r = 0.0;
  delta_s = delta_theta = theta = 0.0;
  v = w = 0.0;
  step_time = 0.0;

  currTime = ros::Time::now().toSec();
  double diff_time = currTime - lastTime;
  step_time = diff_time;

  if (step_time == 0)
    return false;

  wheel_l = (currJointState.position[0] - lastJointState.position[0]);
  wheel_r = (currJointState.position[1] - lastJointState.position[1]);

  if (isnan(wheel_l))
    wheel_l = 0.0;

  if (isnan(wheel_r))
    wheel_r = 0.0;

  delta_s    = WHEEL_RADIUS * (wheel_r + wheel_l) / 2.0;
  q = imu.orientation;
  theta = tf::getYaw(q);
  delta_theta = theta - last_theta;

  // compute odometric pose
  odom_pose[0] += delta_s * cos(odom_pose[2] + (delta_theta / 2.0));
  odom_pose[1] += delta_s * sin(odom_pose[2] + (delta_theta / 2.0));
  odom_pose[2] += delta_theta;
  v = delta_s / step_time;
  w = delta_theta / step_time;

  odom_vel[0] = v;
  odom_vel[1] = 0.0;
  odom_vel[2] = w;

  last_theta = theta;
  lastTime = currTime;
  return true;
}

void updateOdometry()
{
  odom.header.frame_id = odom_header_frame_id;
  odom.child_frame_id = odom_child_frame_id;

  odom.pose.pose.position.x = odom_pose[0];
  odom.pose.pose.position.y = odom_pose[1];
  odom.pose.pose.position.z = 0;
  odom.pose.pose.orientation = tf::createQuaternionMsgFromYaw(odom_pose[2]);

  odom.twist.twist.linear.x = odom_vel[0];
  odom.twist.twist.angular.z = odom_vel[2];
}
```

```
void updateTF()
{
  odom_tf.header = odom.header;
  odom_tf.child_frame_id = odom.child_frame_id;
  odom_tf.transform.translation.x = odom.pose.pose.position.x;
  odom_tf.transform.translation.y = odom.pose.pose.position.y;
  odom_tf.transform.translation.z = odom.pose.pose.position.z;
  odom_tf.transform.rotation   = odom.pose.pose.orientation;
}

void jointStateCallback(const sensor_msgs::JointState::ConstPtr& msg)
{
  static tf::TransformBroadcaster br;
  currJointState = *msg;
  calcOdometry();
  updateOdometry();
  updateTF();
  odom.header.stamp = ros::Time::now();
  odom_pub.publish(odom);
  odom_tf.header.stamp = ros::Time::now();
  br.sendTransform(odom_tf);
  lastJointState = currJointState;

}
void imuCallback(const sensor_msgs::Imu::ConstPtr& msg)
{
  imu = *msg;
}

void init(ros::NodeHandle n)
{
  odom.pose.pose.position.x = 0.0;
  odom.pose.pose.position.y = 0.0;
  odom.pose.pose.position.z = 0.0;

  odom.pose.pose.orientation.x = 0.0;
  odom.pose.pose.orientation.y = 0.0;
  odom.pose.pose.orientation.z = 0.0;
  odom.pose.pose.orientation.w = 0.0;

  odom.twist.twist.linear.x = 0.0;
  odom.twist.twist.angular.z = 0.0;
  std::string get_tf_prefix;
  n.getParam("tf_prefix", get_tf_prefix);
  if (get_tf_prefix == "")
  {
    odom_header_frame_id = "odom";
    odom_child_frame_id = "base_footprint";
  }
  else
  {
    odom_header_frame_id = get_tf_prefix + "/odom";
    odom_child_frame_id = get_tf_prefix + "/base_footprint";

  }
}

int main(int argc, char** argv){
  ros::init(argc, argv, "sim_odom_pub");
  ros::NodeHandle n;
  init(n);
  ros::Subscriber sub = n.subscribe("joint_states", 100 ,jointStateCallback);
```

```
  ros::Subscriber subImu = n.subscribe("imu", 100 , imuCallback);
  odom_pub = n.advertise<nav_msgs::Odometry>("odom", 50);
  // cout << "init fin" << endl;
  ros::spin();
}
```

This program is adapted from the Odometry publisher on the microcontroller. It takes the information from the simulated IMU and joint states to generate Odometry messages for simulation. The orientation is taken from the imu and the position is taken from the joint states. It also publishes the transformations between the base frame and the Odometry frame. The covariance matrix in the Odometry messgae is ignored and its elements are all 0s.

To control the left and right wheel joint using effort command in the simulation, the URDF and Gazebo descriptions of the robot are also modified as follow.

src/turtlebot3/turtlebot3_description/urdf/turtlebot3_waffle_pi_effort_controller.urdf.xacro:

```
<transmission name="tran1">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="wheel_left_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor1">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="tran2">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="wheel_right_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="motor2">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

src/turtlebot3/turtlebot3_description/urdf/turtlebot3_waffle_pi_effort_controller.gazebo.xacro:

```
<gazebo reference="wheel_left_link">
  <mu1>0.9</mu1>
  <mu2>0.9</mu2>
  <kp>500000.0</kp>
  <kd>10.0</kd>
  <minDepth>0.001</minDepth>
  <maxVel>0.1</maxVel>
  <fdir1>1 0 0</fdir1>
  <material>Gazebo/FlatBlack</material>
</gazebo>

<gazebo reference="wheel_right_link">
  <mu1>0.9</mu1>
  <mu2>0.9</mu2>
  <kp>500000.0</kp>
  <kd>10.0</kd>
```

```
  <minDepth>0.001</minDepth>
  <maxVel>0.1</maxVel>
  <fdir1>1 0 0</fdir1>
  <material>Gazebo/FlatBlack</material>
</gazebo>
```

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
  </plugin>
</gazebo>

<!-- <gazebo>
  <plugin name="turtlebot3_waffle_pi_controller" filename="libgazebo_ros_diff_drive.so">
    <commandTopic>cmd_vel_fake</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometrySource>world</odometrySource>
    <publishOdomTF>true</publishOdomTF>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <publishWheelTF>false</publishWheelTF>
    <publishTf>true</publishTf>
    <publishWheelJointState>false</publishWheelJointState>
    <legacyMode>false</legacyMode>
    <updateRate>30</updateRate>
    <leftJoint>wheel_left_joint</leftJoint>
    <rightJoint>wheel_right_joint</rightJoint>
    <wheelSeparation>0.287</wheelSeparation>
    <wheelDiameter>0.066</wheelDiameter>
    <wheelAcceleration>1</wheelAcceleration>
    <wheelTorque>10</wheelTorque>
    <rosDebugLevel>na</rosDebugLevel>
  </plugin>
</gazebo> -->
```

A configuration file was added to make commanding effort to left and right wheel joints in Gazebo simulation possible

/src/ros_control/params/bot_control.yaml

4 Bring up the Turtlebot

Once the remote PC and Raspberry PI are setup according to the instructions provieded in the previous sections, Follow the instructions below to bring up turtlebot.
On Remote PC

```
roscore
```

On SBC

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

On Remote PC, once you followed the instructions for building the workspace and setting up the environment variables, load the environment variables related to this workspace (Note: This has to be done everytime you want to run something from this workspace from a new termianl)

```
cd ASCL_turtlebot3/
source devel/setup.bash
```

To run teleop to control the Turtlebot with keyboard.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

To run the navigation with PID velocity effort controller and SLAM (with localizatin provided by SLAM and no initial map required) on the physical robot.

```
roslaunch turtlebot3_bringup turtlebot3_physical_nav_control.launch
```

To run the navigation with PID velocity effort controller and SLAM (with localizatin provided by SLAM and no initial map required) on Gazebo for simulation.

```
roslaunch turtlebot3_bringup turtlebot3_sim_nav_control.launch
```

To run SLAM only to draw maps Choose a pre-defined Gazebo world from the following:

- turtlebot3_empty_world.launch
- turtlebot3_world.launch
- turtlebot3_house.launch
- turtlebot3_stage_1.launch
- turtlebot3_stage_2.launch
- turtlebot3_stage_3.launch
- turtlebot3_stage_4.launch

For example, if **turtlebot3_world.launch** is chosen, run the following command to bring up a virtual Turtlebot on Gazebo for simulation

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Then, run the following commands to launch the SLAM nodes and remote control using keyboard.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Once you are satisfied with the map, save the map by running

```
rosrun map_server map_saver -f ~/map
```

5 Syncing Time using NTP Server and Client

Raspberry Pi does not have a hardware clock, and therefore it relies on an active internet connection to fetch the accurate time from Ubuntu time server. However, this is not possible as Linksys router, which the raspberry pi will connect to over WiFi, will not have internet access.

Every time pi boots up, the time has to be updated. The time on the Remote PC and the Raspeberry Pi have to be synced up in order for the various transformations to work for the purpose of running SLAM and Navigation in the ACSL Lab Environment. Hence, to sync up time between these two machines, NTP protocol is being used.

NTP or Network Time Protocol is a protocol that is used to synchronize all system clocks in a network to use the same time. When we use the term NTP, we are referring to the protocol itself and also the client and server programs running on the networked computers. NTP belongs to the traditional TCP/IP protocol suite and can easily be classified as one of its oldest parts.

When you are initially setting up the clock, it takes six exchanges within 5 to 10 minutes before the clock is set up. Once the clocks in a network are synchronized, the client(s) update their clocks with the server once every 10 minutes. This is usually done through a single exchange of message(transaction). These transactions use port number 123 of your system.

In this section, a step-by-step procedure is described on how to:

- Install and configure the NTP server on a Ubuntu machine.
- Configure the NTP Client to be time synced with the server.

5.1 Install and configure NTP Server on the host computer

Open Terminal and run the following commands

```
sudo apt-get update
sudo apt-get install ntp
```

Most likely ntp will already be installed on the machine. After installation, the configuration file needs to be altered and server pools closest to the PC location needs to be added. This step is optional, as most likely the servers will be configured correctly.

```
sudo gedit /etc/ntp.conf
```

Replace the server pools with the following.

```
server 0.us.pool.ntp.org
    server 1.us.pool.ntp.org
    server 2.us.pool.ntp.org
    server 3.us.pool.ntp.org
```

Save and close the configuration file. In order for the above changes to take effect, you need to restart the NTP server.

```
sudo service ntp restart
sudo service ntp status
```

Finally, the system's UFW firewall needs to be configured so that incoming connections can access the NTP server at UDP Port number 123.

```
sudo ufw allow from any to any port 123 proto udp
```

Now the Ubuntu host machine is configured to be used as an NTP server.

5.2 Configure NTP Client to be Time Synced with the NTP Server

The ntpdate command will let you manually check your connection configuration with the NTP-server. Only in this case, the server would be the one that we specify. It should already be installed on the SBC, but in case it is not, we need to run the following command.

```
sudo apt-get install ntpdate
```

As the SBC will be a client to the host we set up on the remote PC, we need to modify the hosts file for NTP.

```
sudo gedit /etc/hosts
```

Now add your NTP server's IP and specify a hostname as follows in this file:

```
192.168.1.183 ACSLServer
```

Save and close the NTP hosts file. Running the following command will show the offset between server and client times.

```
sudo ntpdate ACSLServer
```

First we need to disable time sync on the client

```
systemctl stop systemd-timesyncd
systemctl disable systemd-timesyncd
```

Now to correct this time offset we need to configure NTP on the client machine.
If ntp is not installed install it using the following command.

```
sudo apt-get install ntp
```

Now we want our client machine to use our own NTP host server to be used as the default time server. For this, we need to edit the /etc/ntp.conf file on the client machine.

```
sudo nano /etc/ntp.conf
```

Add the following line to the list of servers

```
server ACSLServer prefer iburst
```

Finally restart the ntp service and the time should be synced.

```
sudo service ntp restart
```

Now your client and server machines are configured to be time-synced. You can view the time synchronization queue by running the following command:

```
ntpq -ps
```

6 List of ROS Nodes, Topics, Parameters, Transform

6.1 List of Important ROS Nodes

- /differential_driver
- /left_wheel/pid_left_node
- /message_redirect
- /move_base
- /odom_pub
- /right_wheel/pid_right_node
- /robot_state_publisher
- /rviz
- /turtlebot3_slam_gmapping
- /follow_waypoints

6.2 List of Topics and Their Types

- /map_metadata [nav_msgs/MapMetaData]
- /camera/rgb/image_raw [sensor_msgs/Image]
- /move_base/current_goal [geometry_msgs/PoseStamped]
- /move_base/local_costmap/footprint [geometry_msgs/PolygonStamped]
- /tf [tf2_msgs/TFMessage]
- /move_base/local_costmap/costmap [nav_msgs/OccupancyGrid]
- /odom [nav_msgs/Odometry]
- /joint_left_effort_controller/command [std_msgs/Float64]
- /scan [sensor_msgs/LaserScan]
- /left_vel_fb [std_msgs/Float64]
- /camera/rgb/image_raw/compressed [sensor_msgs/CompressedImage]
- /left_wheel/pid_left_node/parameter_updates [dynamic_reconfigure/Config]
- /move_base_simple/goal [geometry_msgs/PoseStamped]
- /move_base/DWAPlannerROS/cost_cloud [sensor_msgs/PointCloud2]
- /move_base/DWAPlannerROS/trajectory_cloud [sensor_msgs/PointCloud2]
- /tf_static [tf2_msgs/TFMessage]
- /imu [sensor_msgs/Imu]
- /map [nav_msgs/OccupancyGrid]
- /move_base/global_costmap/footprint [geometry_msgs/PolygonStamped]
- /joint_right_effort_controller/command [std_msgs/Float64]
- /cmd_vel [geometry_msgs/Twist]

- /move_base/DWAPlannerROS/local_plan [nav_msgs/Path]
- /waypoints [geometry_msgs/PoseArray]
- /joint_states [sensor_msgs/JointState]
- /move_base/goal [move_base_msgs/MoveBaseActionGoal]
- /initialpose [geometry_msgs/PoseWithCovarianceStamped]
- /rosout_agg [rosgraph_msgs/Log]
- /move_base/GlobalPlanner/plan [nav_msgs/Path]
- /move_base/DWAPlannerROS/global_plan [nav_msgs/Path]
- /left_vel [std_msgs/Float64]
- /right_wheel/pid_debug [std_msgs/Float64MultiArray]
- /camera/rgb/image_raw/compressedDepth [sensor_msgs/CompressedImage]
- /camera/rgb/camera_info [sensor_msgs/CameraInfo]
- /camera/rgb/image_raw/theora [theora_image_transport/Packet]
- /gazebo/link_states [gazebo_msgs/LinkStates]
- /move_base/GlobalPlanner/potential [nav_msgs/OccupancyGrid]
- /gazebo/model_states [gazebo_msgs/ModelStates]
- /clock [rosgraph_msgs/Clock]
- /right_vel_fb [std_msgs/Float64]
- /move_base/cancel [actionlib_msgs/GoalID]
- /right_vel [std_msgs/Float64]
- /move_base/global_costmap/costmap [nav_msgs/OccupancyGrid]
- /left_torque [std_msgs/Float64]
- /right_torque [std_msgs/Float64]
- /battery_state [sensor_msgs/BatteryState]

6.3 List of Important Topics and Descriptions

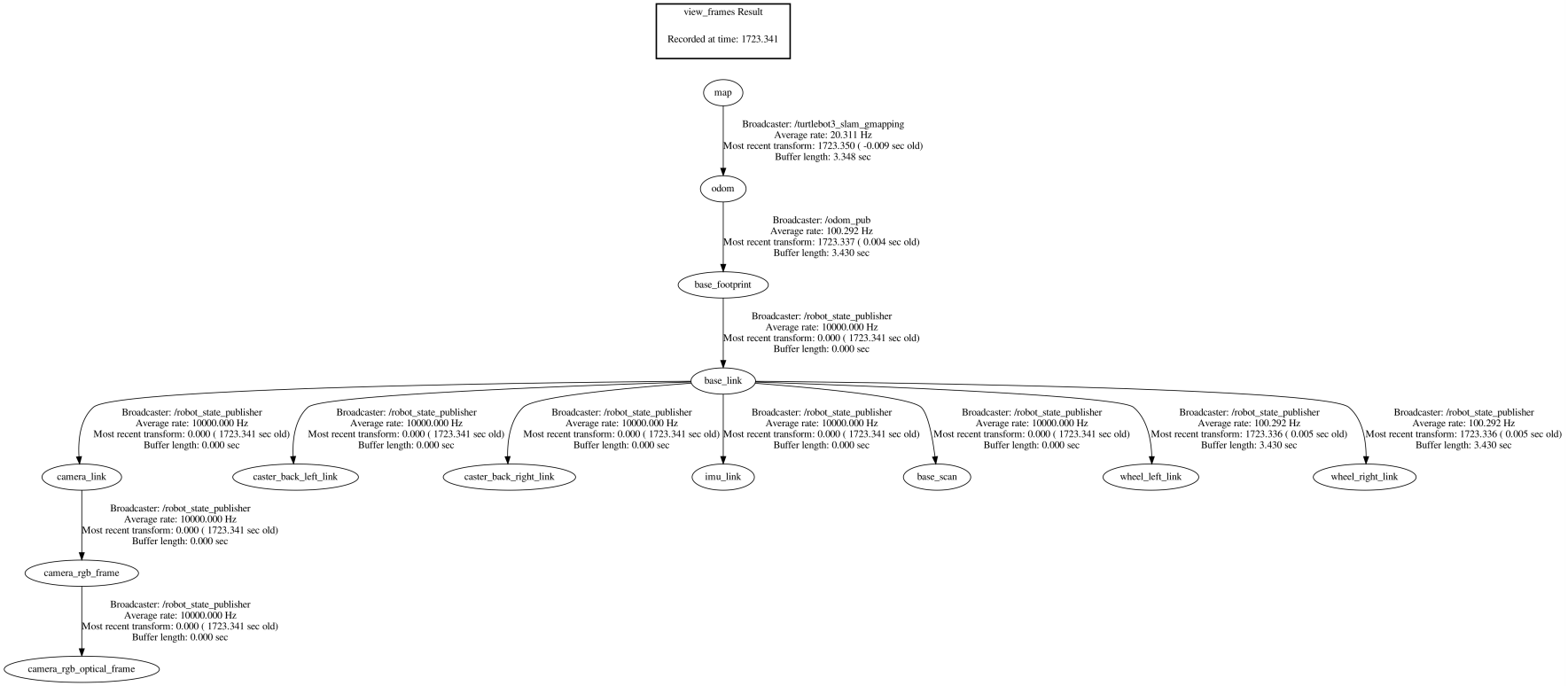
6.4 List of Important Parameters and How to Find Them

Topic Name	Index
/camera/rgb/camera_info	1
/camera/rgb/image_raw	2
/camera/rgb/image_raw/compressed	3
/camera/rgb/image_raw/theora	4
/cmd_vel	5
/imu	6
/initialpose	7
/joint_left_effort_controller/command	8
/joint_right_effort_controller/command	9
/joint_states	10
/left_vel	11
/left_vel_fb	12
/left_wheel/pid_enable	13
/map	14
/map_metadata	15
/move_base/DWAPlannerROS/local_plan	16
/move_base/GlobalPlanner/plan	17
/move_base/global_costmap/costmap	18
/move_base/goal	19
/move_base/local_costmap/costmap	20
/odom	21
/path_ready	22
/path_reset	23
/right_vel	24
/right_vel_fb	25
/right_wheel/pid_enable	26
/scan	27
/waypoints	28
/left_torque	29
/right_torque	30
/battery_state	31

Index	Descriptions
1	http://docs.ros.org/melodic/api/sensor_msgs/html/msg/CameraInfo.html
2	Raw camera data directly from sensor
3	Compressed image
4	Compressed image based on therora encoding
5	Command Velocity from ROS navigation stack
6	IMU Information (acceleration, orientation)
7	Originally initial pose estimate of AMCL, instead used for adding waypoints
8	Left effort command (only in simmulation)
9	Right effort command (only in simmulation)
10	Left and right wheel joint states (angular velocity, angular position)
11	Left wheel velocity command
12	Left wheel velocity feedback
13	Left wheel PID controller enable
14	Map drawn by SLAM algorithm
15	Map mata information (width, height, resoulution)
16	Local trajectory to follow by robot generated by the local planner (uses odom frame)
17	Global plan generated by the global planner
18	Global costmap generated from SLAM map and lidar (uses map frame)
19	Current goal to reach
20	Local costmap generated from part of the global costmap and lidar (uses odom frame)
21	Odometry of the robot (accumulated orientation and position based on the feedabck of the motor and IMU)
22	Singal to start following waypoints
23	Singla to reset the waypoints
24	Right wheel velocity command
25	Right wheel velocity feedback
26	Right wheel PID controller enable
27	Data from lidar (in terms of angle and distance)
28	Array of waypoints for visualization purpose
29	Left torque command (only in physical robot)
30	Right torque command (only in physical robot)
31	Battery voltage, designed capacity, percentage left (only in physical robot)

Parameters	Location
Kp, Ki, Kd in simulation	src/ros_control/launch/turtlebot3_control.launch
Kp, Ki, Kd in physical robot	src/ros_control/launch/turtlebot3_control_simulation.launch
Local planner parameters	src/turtlebot3/turtlebot3_navigation/ param/dwa_local_planner_params_waffle_pi_effort_controller.yaml
Global planner parameters	src/turtlebot3/turtlebot3_navigation/param/global_planner_params.yaml
move_base parameters	src/turtlebot3/turtlebot3_navigation/param/move_base_params.yaml
Cartographer parameters in physical robot	src/turtlebot3/turtlebot3_slam/config/turtlebot3_lds_2d.lua
Cartographer parameters in simulation	src/turtlebot3/turtlebot3_slam/config/turtlebot3_lds_2d_gazebo.lua
Hector SLAM parameters	src/turtlebot3/turtlebot3_slam/launch/turtlebot3_hector.launch
AMCL parameters	src/turtlebot3/turtlebot3_navigation/launch/amcl.launch
Gmapping parameters	src/turtlebot3/turtlebot3_slam/config/gmapping_params.yaml
Common costmap parameters	src/turtlebot3/turtlebot3_navigation/param/costmap_common_params_waffle_pi_
Global costmap parameters	src/turtlebot3/turtlebot3_navigation/param/global_costmap_params.yaml
Local costmap parameters	src/turtlebot3/turtlebot3_navigation/param/local_costmap_params.yaml
SLAM Method	src/turtlebot3/turtlebot3_navigation/launch/turtlebot3_navigation_no_map.launc

/right_vel
/left_vel
/differential_driver
/cmd_vel
/move_base
/move_base/global_costmap
/move_base/global_costmap/footprint
/move_base/global_costmap/costmap_updates
/move_base/global_costmap/costmap
/move_base/local_costmap
/move_base/local_costmap/costmap_updates
/move_base/local_costmap/costmap
/move_base/local_costmap/footprint
/move_base/DWAPlannerROS
/move_base/DWAPlannerROS/global_plan
/move_base/DWAPlannerROS/local_plan
/move_base/action_topics
/move_base
/follow_waypoints
/path_reset
/start_journey
/initialpose
/path_ready
/move_base_simple
/move_base_simple/goal
/map
/right_wheel
/right_wheel/pid_right_node
/right_wheel/pid_enable
/right_vel_fb
/scan
/turtlebot3_slam_gmapping
/odom
/tf
n__rviz
/left_wheel
/left_wheel/pid_left_node
/left_wheel/pid_enable
/left_vel_fb
/odom_pub
/message_redirect
/robot_state_publisher
/imu
/joint_states
/gazebo
/gazebo
/gazebo/set_link_state
/gazebo/set_model_state
/joint_right_effort_controller
/joint_right_effort_controller/command
/joint_left_effort_controller
/joint_left_effort_controller/command
/controller_spawner
/gazebo_gui


view_frames Result
Recorded at time: 1723.341
map
Broadcaster: /turtlebot3_slam_gmapping
Average rate: 20.311 Hz
Most recent transform: 1723.350 (-0.009 sec old)
Buffer length: 3.348 sec
odom
Broadcaster: /odom_pub
Average rate: 100.292 Hz
Most recent transform: 1723.337 (0.004 sec old)
Buffer length: 3.430 sec
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
camera_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
caster_back_left_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
caster_back_right_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
imu_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
base_scan
Broadcaster: /robot_state_publisher
Average rate: 100.292 Hz
Most recent transform: 1723.336 (0.005 sec old)
Buffer length: 3.430 sec
wheel_left_link
Broadcaster: /robot_state_publisher
Average rate: 100.292 Hz
Most recent transform: 1723.336 (0.005 sec old)
Buffer length: 3.430 sec
wheel_right_link
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
camera_rgb_frame
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1723.341 sec old)
Buffer length: 0.000 sec
camera_rgb_optical_frame